Panel: Practical Issues in Implementing Software Reliability Measurement

Chair: Allen P. Nikora, Jet Propulsion Laboratory, California Institute of Technology
Panelists: John D. Musa, Software Reliability Engineering and Testing Courses
Norman F. Schneidewind, Division of Computer and Information Sciences and Operations, Naval Postgraduate School
William W. Everett, SPRE, Inc.
John C. Munson, Computer Science Department, University of Idaho
Mladen A. Vouk, Department of Computer Science, College of Engineering, North Carolina State University

Abstract

Many ways of estimating software systems' reliability, or reliability-related quantities, have been developed over the past several years. Of particular interest are methods that can be used to estimate a software system's reliability or fault content prior to test, or to discriminate between components that are fault-prone and those that are not. In this panel, we discuss practical issues to be addressed in implementing software reliability measurement techniques in a production development environment.

Statement of Bill Everett

As indicated in the Panel Statement, there is interest in developing methods that can be applied earlier in the development cycle (before system testing) to estimate software reliability. Estimating software fault content or fault-proneness is important as software's propensity to fail is strongly correlated to the remaining faults in the software.

However, we also need to include "dynamic" properties of how the software is to be used to develop a fuller picture of software's failure-proneness. Indeed, a part of the software may have a large number of faults, but if it is never exercised through the usage we expect to put it through, then those faults will never trigger failures.

To develop a more complete picture, we also have to estimate dynamic properties of the software. These include where the processing will occur in the software and how much processing will occur (these are correlated to the "operational profile"). These dynamic properties determine the likelihood of tripping over a fault. In addition, we need to estimate the likelihood that a fault will trigger a failure when it is encountered (fault exposure ratio).

Also, we need to develop these estimates on a software component basis rather than just a system basis. Given fault content estimates and the estimates of the dynamic properties of software components, we can then develop estimates of reliability indicators on a component basis. This should be done early in the software development life-cycle to influence and guide the design of the software to meet reliability objectives. Estimates can be refined during development as more accurate information on individual components becomes available.

For safety-critical systems, these software reliability indicators can be used along with other safety assessment techniques such as fault tree analysis to develop probabilistic risk estimates of critical failure events. Doing such analysis early in the development life-cycle provides insight into which components contribute most to the risk of occurrence of such events and allows the opportunity of introducing "mitigations" to reduce their contribution.

Also, the reliability indicators can be used to develop (and calibrate) software reliability growth models of both individual components and the software system. Failure data collected during system test can then be used to validate the models and hence the underlying reliability of the system.

Statement of John Musa

The principal practical issues in measuring reliability include determining when the failures occurred, dealing with small failure samples, making test represent the field, and handling system evolution.

"When the failure occurred" must be expressed in terms that relate to the amount of processing. The simplest way to do this is to use clock time, but this is satisfactory only if there is constant average (over a length of time roughly equal to a failure interval) computer utilization. Using execution time (the actual time instructions are executing) is precise, but it is difficult to instrument for distributed systems, and these are very common these days. Natural units, which are measures related to the output of a software-based product, such as pages of output, transactions, or telephone calls, generally provides the best solution. They are not always common across all operations, but this problem can be handled by choosing a reference natural unit and converting other natural units to that reference.

In many cases, particularly when reliability is high, you may only have a small sample of failures. In this situation, estimates by severity class, operation, or component may be of limited value because of large confidence intervals. If the proportions among the different groups of failures are stable, you can use all failures to estimate total failure intensity and then apply the proportions to obtain the group failure intensities.

Often when you are making reliability measurements, you are doing so in test with the goal of estimating the reliability you will obtain in the field. These estimates are only valid if test represents the field. Hence you must develop the operational profile to characterize use in the field, so that you can accurately reproduce it in test. You randomly select tests, at all times using the same operational profile (don't rearrange the order of tests). Use the same data base cleanup procedure in test as you will in the field. You need to recognize that feature and regression tests will somewhat underestimate failure intensity, because failures that result from interaction of operations and data degradation will not be present. In many cases, however, load test will predominate over feature and regression tests, so any error in making reliability measurements will be small.

The other major issue is the system evolution that can occur during test. The most practical way to handle this is to measure failure intensities of operation groups or components separately. These are then combined when the evolution causes the operation groups or components to be combined. Again, failure intensity may be underestimated because failures resulting from interactions may not occur.

Statement of Norm Schneidewind

While software design and code metrics have enjoyed some success as predictors of software quality attributes such as reliability , the measurement field is stuck at this level of achievement. If measurement is to advance to a higher level, we must shift our attention to the front-end of the development process, because it is during system conceptualization that errors in specifying requirements are inserted into the process. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors propagate through later phases of development and maintenance. These errors may result in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements). Although requirements may be specified correctly in terms of meeting user expectations, there could be significant risks associated with their implementation. For example, correctly implementing user requirements could lead to excessive system size and complexity with adverse effects on reliability or there could be a demand for project resources that exceeds the available funds, time, and personnel skills. Interestingly, there has been considerable discussion of project risk (e.g., the consequences of cost overrun and schedule slippage) in the literature but not a corresponding attention to reliability risk.

The generation of requirements is not a one-time activity. Indeed, changes to requirements can occur during maintenance. When new software is developed or existing software is changed in response to new and changed requirements, respectively, there is the potential to incur reliability risks. Therefore, in assessing the effects of requirements on reliability, we should deal with changes in requirements throughout the life cycle. In addition to the relationship between requirements and reliability, there are the intermediate relationships between requirements and complexity and between complexity and reliability. These relationships may interact to put the reliability of the software at risk because the requirements changes may result in increases in the size and complexity of the software that may adversely affect reliability.